Design Document
Final Project - CS5610

Abhimanyu Kumar, Yunfei Zhou

Instructions for local setup/load (including .env file setup)

1. Git clone the repository locally.
2. Get your AI API keys - used to prompt the LLMs as part of the application.
   a. Groq: https://console.groq.com/keys
   b. Gemini: https://aistudio.google.com/apikey
3. Set up backend
   a. *cd server*
   b. *cp .env-example .env* (this creates .env file as a copy of the .env-example for you)
   c. Add your API keys to your .env file:
      i. GROQ_API_KEY= “your GROQ API key from step 2a”
      ii. GEMINI_API_KEY= “your Gemini API key from step 2b”
   d. Set up your PostgreSQL credentials (to utilise the PostgreSQL’s persistent database) in the .env file.
      i. DATABASE_URL=
      ii. DB_NAME=
      iii. DB_USER=
      iv. DB_PASSWORD=
      v. DB_HOST=
      vi. DB_PORT=
         1. Note: IF POSTGRESQL CREDENTIALS ARE NOT SETUP, THE APPLICATION RUNS IS STILL FUNCTIONAL AND RUNS ON **SQLITE** AS A BACKUP DATABASE CONFIGURATION - an extra functionality/feature of the application.
   e. Set up your Python virtual environment (mac commands below):
      i. python3 -m venv venv
      ii. source venv/bin/activate
      iii. pip install -r requirements.txt
         1. Note: If pip does not work, replace pip with pip3 as your system might be configured to identify pip3 for installing requirements.
      iv. python manage.py migrate
   f. Start the backend with this command: *./backend.sh*
      i. Now the backend should be serving on http://localhost:3001 or http://127.0.0.1:3001
4. Set up frontend
   a. *cd client*

   b. *cp .env-example .env* (this creates .env file as a copy of the .env-example for you)
      i. In .env, set up:
         1. VITE_API_URL= “your backend API (locally, it should be http://localhost:3001)”
   c. *npm install* (to install all dependencies)
   d. *npm run dev* (to run the application)
      i. Now the frontend should be serving on http://localhost:5173 and you can open that link to run the application.

List of technologies used:

a. Frontend
   i. React 19.2.0
   ii. Vite 7.2.2
b. Backend
   i. Django 4.2.7
   ii. Python 3.8+
   iii. Groq API (Llama 3.3 70B)
   iv. Google Generative AI (Gemini)
c. Database
   i. PostgreSQL
   ii. SQLite (added as an extra feature of the application, as the backup database)

USER PERSONAS

1. AI User (Primary User)

- Wants accurate and fast comparison between AI responses
- Uses history to review past tests
- Prefers customizable system prompts
- Values clarity and structured evaluation

2. Prompt Engineer / Power User

- Creates many prompts
- Needs side-by-side comparison and customizable system context
- Wants features like saved prompts, rubrics, scoring

3. Instructor / Reviewer

- Uses rubric scoring to evaluate AI responses
- Needs saved comparisons for teaching
- Wants organized query history

USER STORIES

Authentication

- As a user, I want to sign up with my email so that I can securely access the app.
- As a user, I want to log in (through a safe method like JWT) so that I can keep my session secure.

Profile

- As a user, I want to update my profile, name, and bio so that my account feels personalized.
- As a user, I want to manage my account settings and delete my account if needed.

AI Comparison

- As a user, I want to submit a prompt and compare responses from Groq and Gemini.
- As a user, I want to optionally add a system prompt/context.
- As a user, I want to visually compare responses side-by-side.

Evaluation

- As a user, I want an AI-generated rubric evaluation for each model so I understand which response is better.

History

- As a user, I want to revisit past queries and view details.

DATABASE SCHEMA

USER TABLE

Table: users_user

Column	Type	Constraints
id	bigint	PK, auto-increment
email	varchar(254)	UNIQUE, NOT NULL
password	varchar(128)	NOT NULL
last_login	datetime	NULL
is_active	boolean	DEFAULT TRUE
is_staff	boolean	DEFAULT FALSE
is_superuser	boolean	DEFAULT FALSE
date_joined	datetime	DEFAULT now
username	varchar(150)	NULL
first_name	varchar(30)	NULL
last_name	varchar(30)	NULL
phone	varchar(15)	NULL
location	varchar(200)	NULL
bio	text	NULL

Relationships:

- Inherits permissions system from Django:

    - users_user_groups table
    - users_user_user_permissions table

QUERY HISTORY TABLE

Table: yourappname_queryhistory

Column	Type	Constraints
id	bigint	PK
user_id	bigint	FK → users_user(id) ON DELETE CASCADE
prompt	text	NOT NULL
response_groq	text	NULL
response_gemini	text	NULL
mode	varchar (20)	DEFAULT 'both'
created_at	datetime	auto_now_add

Constraints:

- Ordered by created_at DESC

Base URL

/api/ (in each endpoint, it begins with /api, added it here to remove repetition below)

Health Check

Endpoint: /health/
Method: GET
Description: Check if the API is running.

Response:

```
{
  "status": "ok",
  "message": "AI Comparator API is running"
}
```

AI Model Endpoints

Groq

Endpoint: /groq/
Method: POST
Body Parameters:

```
{
  "prompt": "string"
}
```

Description: Returns response from Groq model. Saves query to history if user is authenticated.

Responses:

- 200 OK: Successful response

```
{
  "model": "Groq",
  "response": "AI-generated text",
  "timestamp": "ISO timestamp"
}
```

- 400 Bad Request: Prompt missing

```
{ "error": "Prompt is required" }
```

- 500 Internal Server Error: API failure

Gemini

Endpoint: /gemini/
Method: POST
Body Parameters: Same as Groq

Responses: Same format as Groq, but model is "Gemini".

Compare (Both Models)

Endpoint: /compare/
Method: POST
Body Parameters: Same as above

Description: Gets responses from both Groq and Gemini, saves to history if user is authenticated.

Response:

```
{
  "groq": { "model": "Groq", "response": "...", "timestamp": "..." },
  "gemini": { "model": "Gemini", "response": "...", "timestamp": "..." }
}
```

Compare with Rubric

Endpoint: /compare-with-rubric/
Method: POST
Body Parameters: Same as above

Description: Returns AI responses from both models and an evaluation based on a rubric (accuracy, relevance, clarity, completeness, usefulness).

Response Example:

```
{
  "prompt": "string",
  "responses": {
    "groq": { "model": "Groq", "response": "...", "timestamp": "..." },
    "gemini": { "model": "Gemini", "response": "...", "timestamp": "..." }
  },
  "evaluation": {
    "success": true,
    "rubric": { ... },
    "evaluator": "Gemini Flash"
  }
}
```

Authentication Endpoints

Register

Endpoint: /auth/register/
Method: POST
Body Parameters:

```
{
  "email": "string",
  "password": "string",
  "username": "string"
}
```

Response:

```
{
  "message": "User registered successfully",
  "user": { "id": 1, "email": "...", "username": "..." },
  "tokens": { "access": "...", "refresh": "..." }
}
```

Login

Endpoint: /auth/login/
Method: POST
Body Parameters:

```
{
  "email": "string",
  "password": "string"
}
```

Response: Same format as register.

Get Current User

Endpoint: /auth/user/
Method: GET
Headers: Authorization: Bearer <access_token>

Response:

```
{
  "user": { "id": 1, "email": "..." }
}
```

User Query History

Endpoint: /history/
Method: GET
Headers: Authorization: Bearer <access_token>

Description: Returns last 5 queries of the authenticated user.

Response Example:

```
{
  "history": [
    {
      "id": 1,
      "prompt": "...",
      "mode": "groq/gemini/both",
      "created_at": "ISO timestamp",
      "responses": { "groq": "...", "gemini": "..." }
    }
  ]
}
```

User Profile

Endpoint: /profile/
Method: GET, PUT, DELETE
Headers: Authorization: Bearer <access_token>

GET: Returns user profile information.

PUT: Update profile fields. Body can include:

```
{
  "username": "string",
  "first_name": "string",
  "last_name": "string",
  "phone": "string",
  "location": "string",
  "bio": "string"
}
```

DELETE: Deletes user account.

Responses:

- Success: JSON with updated profile or deleted account email
- Error: JSON with error message